C13

```
/*
** Copyright 1996,1997 EMC Corporation
*/

/*
** EDMDispatch.c
**
** Mission Statement: This is the main service file for the dispatch
                                                             daemon.
**                    This file contains the callbacks from the main
                                                           function
**                    which prepares the daemon to go off and service
                                                              RPC's.
**
** Primary Data Acted On:
**
** Compile-Time Options:
**
**                   None.
**
** Basic idea here: Module for UNIX specific daemon initialization
*/

/*
** The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/
#if !defined(lint)
static char     RCS_id [] = "@(#)$RCSfile: EDMDispatch.c,v $ "
                            "$Revision: 1.23 $ "
                            "$Date: 1997/02/06 20:49:15 $" ;
#endif

/* #define _POSIX_SOURCE   unable to compile with this define set */
/* #define _XOPEN_SOURCE   unable to compile with this define set */

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <esl/inout.h>

#include <stdarg.h>
#include <string.h>
#include <syslog.h>
#include <pthread.h>
#include <thread.h>
#include <sys/resource.h>

#include <logging/logging.h>
#include <util/esl_core.h>
#include <util/esl_pidfile.h>
#include <util/esl_daemon.h>
#include <csc/cscomm.h>

#include <restore/csc_EDMDispatch.h>

#include <EDMmain.h>
#include <EDMDD_ccw.h>
#include <EDMDispatchLog.h>
#include <EDMDispatchBackground.h>
#include <EDMDDcr_rstsvc.h>

/*
 * Need to define _XOPEN_SOURCE for signal funtion definitions
 * and certain signal structure definitions.
 */
#define _XOPEN_SOURCE
```

```
#include <signal.h>

#undef _XOPEN_SOURCE

static rpc_if_handle_t if_spec;

static int       G_debug = FALSE;
                                 /* Variable which will disable forking */

static char      **commandlineargs; /* Pointer to command line args */

/*******************************************************************
                                                            ********
**
** Routine: IsDebugOn
**
** Inputs: None
**
** Outputs:  None
**
** Return Codes:
**              TRUE if debug is on.
**
** Purpose:     This routine can be used to tell other subsystems
**              whether debugging is available.
**
** Intended caller:  internal only.
**
********************************************************************
                                                            ********
*/
boolean_ty
IsDebugOn()
{
    return G_debug;
}
```

```
/*********************************************************************
                                                              ********
**
** Routine: kill_handler
**
** Inputs: int sigval - the signal which was receieved.
**
** Outputs:  Will log messages telling what action is being taken.
**
** Return Codes:
**                exits with the number of the signal received
**
** Purpose:       This routine handles specific signals i.e. SIGINT,
**                SIGQUIT,
                          SIGTERM. Each results in a log entry and an exit.
**
** Intended caller:  internal only.
**
*********************************************************************
                                                              ********
*/
static void kill_handler( IN int sigval )
{
int       status;
time_t    current_time;
char      *ctimebuf;
char      *ebuff;

  /* If main exits, it calls this routine with signal 0 */

  /* Unregister the interface */
  (void) csc_unregister_server_interface(&if_spec, &status);

  /* If the unregister fails, report the problem, but continue */
  if ( status != error_status_ok )
  {
    ebuff = (char *) csc_get_error( status );

    (void) EDMDispatch_logent(
                     __FILE__,__LINE__, LOG_ERR, MESSAGE_NO_LOGIN, 0,
                        "CSC_SERVER_LOGIN failed: <%d> %s",
                        status, (ebuff ? ebuff : "Unknown error") );
  }

  /* Get the current time */
  (void) time(&current_time);

  ctimebuf = ctime(&current_time);

  /* Overlay newline with null - buf should always be 26 bytes long */
  ctimebuf[ strlen(ctimebuf) - 1 ] = 0;

  (void) EDMDispatch_logent(
                     __FILE__,__LINE__, LOG_INFO, MESSAGE_SHUTDOWN, 0,
                     "Shutting down at %s due to signal %d", ctimebuf,
                                                          sigval);

  /*
   * Remove our lock file.
   */
  (void) EslDestroyPidFile(PIDPATH);

  exit(sigval);

} /* End of kill_handler() */
```

```
/**********************************************************************
                                                               ********
 * Function Name:
 *       display_usage
 *
 * Simply displays the usage
 *
 * Call Arguments:
 *       Program name
 *
 * Error Outputs and Side Effects:
 *       Prints usage.
 *
 * Special Considerations:
 *       None.
 *
**********************************************************************
                                                               ********
 */
static void
display_usage (IN char *progname)
{
  /* Print out usage stmt.  */

  fprintf (stderr,"Usage: %s [-d]\n", progname);
  fprintf (
   stderr,"-d keep the daemon from forking so debugging is easier\n");

} /* end display_usage () */
```

```
/***********************************************************************
                                                                ********
**
** Routine: daemon_catch_interrupts
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              None
**
** Purpose:     Sets up signals for service. On NT we will have to
**              consider what OS constructs to replace signals with.
**              In this case we are catching SIGTERM, SIGINT, and
**              SIGQUIT and ignoring anything else.
**
** Intended caller:  internal only.
**
************************************************************************
                                                                ********
*/
void daemon_catch_interrupts()
{
struct sigaction        sactions;       /* Signal actions */

        ZERO( sactions );

        /*
         * Set an empty list so we can set signals we want to handle
         */
        (void) sigemptyset( &sactions.sa_mask );

        /*
         * Add signals that we want to handle
         */
        (void) sigaddset( &sactions.sa_mask, SIGTERM );
        (void) sigaddset( &sactions.sa_mask, SIGINT );
        (void) sigaddset( &sactions.sa_mask, SIGQUIT );

        /* Setup the signal handler. */
        sactions.sa_handler = kill_handler;

        /*
         * Assign handler to each signal we are interested in.
         */
        (void) sigaction( SIGTERM, &sactions, NULL );
        (void) sigaction( SIGINT,  &sactions, NULL );
        (void) sigaction( SIGQUIT, &sactions, NULL );

        /*
         * Setup mask so we can specify what signals we will ignore.
         */
        (void) sigfillset( &sactions.sa_mask );

        /*
         * We want to ignore everything except those we have set up
         * above so remove those from the list.
         */
        (void) sigdelset( &sactions.sa_mask, SIGTERM );
        (void) sigdelset( &sactions.sa_mask, SIGINT );
        (void) sigdelset( &sactions.sa_mask, SIGQUIT );

        /*
```

```
         * Set the mask. Since no other threads have been started,
         * all threads will get this mask.
         */
        (void) thr_sigsetmask( SIG_SETMASK, &sactions.sa_mask, NULL );
}
```

```
/**********************************************************************
                                                              ********
**
** Routine: daemon_check_proper_ID
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              exits with an error when the user is not root
**
** Purpose:     Checks user's ID and determines if the user is allowed
**              to execute service.
                             If there are no constraints then this
**              function may be blank.
**
** Intended caller:  internal only.
**
**********************************************************************
                                                              ********
*/

void daemon_check_proper_ID()
{
        /*
        ** Check for root
        */

        if (geteuid() != E_ROOTUID)
        {
          (void) EDMDispatch_logent(
                  __FILE__,__LINE__, LOG_ERR, DAEMON_NOTSUPERUSER, 0,
                          "Must be run as superuser, uid was %d",
                          geteuid());
          exit(1);
        }
}
```

```
/**********************************************************************
                                                              ********
**
** Routine: parse_commandline
**
** Inputs:      argc, argv (command line arguments)
**
** Outputs:     None
**
** Return Codes:
**              exits with an error when the user types a bad argument
**
** Purpose:     Parses command line arguments and sets flags. If there
**              are no flags to be set then this function may be empty.
**
** Intended caller:  internal only.
**
**********************************************************************
                                                              ********
*/

void parse_commandline(int argc, char *argv[])
{
int                         opt;            /* Process options    */

        commandlineargs = argv;

        while ((opt = getopt(argc,argv,"dD")) != EOF )
        {
          switch(opt)
          {
            case 'd':
            case 'D':
                G_debug = TRUE;
                break;
            default:
                (void) display_usage( argv[0] );
                exit(1);

          }
        }
}
```

```
/*************************************************************************
                                                                  ********
**
** Routine: daemon_initialize_logging
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              None
**
** Purpose:     Do whatever it takes to initialize logging.
                                                           In the near
**              future this may involve doing something with catalogs
                                                                    or
**              calling higher level logging functions which
                                                           encapsulate
**              these things.
**
** Intended caller:  internal only.
**
*************************************************************************
                                                                ********
*/

void
daemon_initialize_logging()
{
        /* Pass in argv[0], the program name */
        (void) esl_log_init(commandlineargs[0]);
}
```

```
/*************************************************************************
                                                                  ********
**
** Routine: daemon_become_daemon
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              exits with an error code if initialization fails
**
** Purpose:     This function is for doing the forking etc. under UNIX.
**              It is unknown what will be necessary under NT.
**
** Intended caller:  internal only.
**
*************************************************************************
                                                                  ********
*/

void
daemon_become_daemon()
{
         char *ptr;
        int  ret = 0;

        /*
         * Strip the path from the program name so we can use it
         * elsewhere.
         */
        ptr = strrchr(commandlineargs[0], '/');
        if (ptr == NULL)
            ptr = commandlineargs[0];
        else
            ptr++;

        /* Change directory to a process specific core directory */
        ret = esl_coredir_setup(ptr);
        if (ret != 0)
        {
             (void) EDMDispatch_logent( __FILE__,__LINE__,LOG_ERR,
                              MESSAGE_ERR_IN_ESL_COREDIR, 0,
                              "esl_coredir_setup failed: <%d>",
                              errno);
            exit(1);
        }

         /*
         **  This is now esl functionality.
                                 This code does everything necessary
         **  to make this a "real" daemon by detaching from the
                                                             terminal
         **  changing the process group, closing stdout, stderr, stdin,
                                                                  ...
         */

        if (G_debug == FALSE)
        {
            ret = esl_daemon_startup();
             if (ret != 0)
            {
                fprintf(
                      stderr, "%s: Failed to initialize as daemon.\n",
                        commandlineargs[0]);
                        exit(1);
                }
        }
}
```

```
/*********************************************************************
                                                              ********
**
** Routine: rpc_init
**
** Inputs:       None
**
** Outputs:      None
**
** Return Codes:
**               exits with an error code if initialization fails
**
** Purpose:      This function is for doing RPC initialization.
**               For the most part it involves calling the csc routines.
**               This is pretty standard between UNIX and NT.
**
** Intended caller:  internal only.
**
*********************************************************************
                                                              ********
*/

void rpc_init()
{
error_status_t           status;
                                             /* error status (nbase.h) */
char   *ebuff;

/*
** This is here because of HP which may or may not define timeval.
** May be removed when esl_timeval is ported to clients
*/
#ifdef _STRUCT_TIMEVAL
        struct timeval  sleep_interval = {5,0};
                                         /* 5 second sleep interval */
#else
        struct timespec sleep_interval = {5,0};
                                         /* 5 second sleep interval */
#endif

        /* Setup the interface specification for RPC */
        SERVER_IFSPEC(if_spec);

        /*
        * Login as SERVER_PRINCIPAL.  The context of the process
        * will be set to this principal.
        *
        * This process will keep trying to login to DCE if the
                                                      registery
        * server is unavailable.
                          Note that under SUN RPC this is a no-op.
        */
        while (TRUE)
        {
                (void) csc_server_login(SERVER_PRINCIPAL,
                                       SERVER_KEYTAB, &status);

                /* If we succeeded, then exit this loop.  */
                if ( status == error_status_ok )
                {
                        break;
                }
                else    /* Print error message if appropriate. */
                {
                 ebuff = (char *) csc_get_error( status );
```

```
                    (void) EDMDispatch_logent(
                                        __FILE__,__LINE__,LOG_ERR,
                                  MESSAGE_NO_LOGIN, 0,
                                  "CSC_SERVER_LOGIN failed: <%d>
                                                          %s",
                                  status, (
                                ebuff ? ebuff : "Unknown error"));
                  }

                  /* If the failure was due to unavailable client,
                  * pause and then try again.
                  */
                  if (status == sec_rgy_server_unavailable)
                  {
                          /*
                           * uses sleep when SUNRPC, otherwise uses
                           * pthread call to delay for the specified
                           * time
                          */
                            CSC_SLEEP(sleep_interval);
                            continue;
                  }

                  /* If we got here, we had a unexpected failure.  */
                  (void) EDMDispatch_logent( __FILE__,__LINE__,LOG_ERR,
                                    MESSAGE_NO_LOGIN, 0,
                                   "The service cannot log in as
                                                         required");

                  exit(1);
          }

          /*
          ** We need to initialize the authorization module before we
                                                                  do
          ** a listen.
          */
          (void)csc_authorization_init(&status);

          if ( status != error_status_ok )
          {
           ebuff = (char *) csc_get_error( status );

             (void) EDMDispatch_logent( __FILE__,__LINE__,LOG_ERR,
                           MESSAGE_NOAUTHORIZATION, 0,
                           "CSC_AUTHORIZATION_INIT failed: <%d> %s",
                            status, (
                                 ebuff ? ebuff : "Unknown error") );
             exit(1);
          }


          (void) csc_register_server_interface(   &if_spec,
                                                  SERVER_ANNOTATION,
                                                  &status);

          if ( status != error_status_ok )
          {
           ebuff = (char *) csc_get_error( status );

             (void) EDMDispatch_logent( __FILE__,__LINE__,LOG_ERR,
                            MESSAGE_CANNOTREGISTER, 0,
                           "CSC_REGISTER_SERVER_INTERFACE failed:
                                                       <%d> %s",
                            status, (
```

```
                                  ebuff ? ebuff : "Unknown error") );
             exit(1);
          }
}
```

```
/*************************************************************************
                                                                ********
**
** Routine: rpc_run
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              None
**
** Purpose:     This function is for running the RPC listen.
**              This is pretty standard between UNIX and NT.
**
** Intended caller:  internal only.
**
*************************************************************************
                                                                ********

*/

void rpc_run()
{
error_status_t                  status;
                                          /* error status (nbase.h) */
char   *ebuff;

            /* listen for RPC calls forever. */
            (void) csc_server_listen(
                              rpc_c_listen_max_calls_default, &status );

          ebuff = (char *) csc_get_error( status );

            /* We don't expect to get here. */
            (void) EDMDispatch_logent(
                     __FILE__,__LINE__,LOG_ERR, MESSAGE_SERVERLISTEN, 0,
                              "CSC_SERVER_LISTEN failed: <%d> %s",
                               status, (
                                    ebuff ? ebuff : "Unknown error") );

}
```

```
/*************************************************************************
                                                                ********
**
** Routine: daemon_specific_initialization
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              None
**
** Purpose:     Do whatever makes this daemon special.
                                                    In some cases you
**              may want to start a thread or open a socket.
                                                        Do that here.
**
** Intended caller:  internal only.
**
*************************************************************************
                                                                ********

*/

void
daemon_specific_initialization()
{
error_status_t              status;
                                           /* error status (nbase.h) */
int                         ret;
pthread_t                   mantid;
pthread_t                   cleantid;
time_t   current_time;
char     *ctimebuf;
struct rlimit rlp;

        /* Create a file and lock it so we don't start multiple
         ** daemons.  Exit if there is another copy of us running.
         ** The CreatePidFile call already logs errors so just exit.
         */
        if (EslCreatePidFile(PIDPATH))
        {
            exit(1);
        }

         /* Find out what time it is */
         (void) time(&current_time);

         ctimebuf = ctime(&current_time);

         /* Overlay newline with null - buf should always be 26 bytes
                                                                long */
         ctimebuf[ strlen(ctimebuf) - 1 ] = 0;

         /* Log startup message */
         (void) EDMDispatch_logent(
                      __FILE__,__LINE__, LOG_INFO, MESSAGE_STARTUP, 0,
                            "Restore service %s starting up at %s",
                                              commandlineargs[0],
                            ctimebuf );

        /* set the open files limit very large */
        rlp.rlim_max = FD_SETSIZE;
        rlp.rlim_cur = FD_SETSIZE;

        setrlimit(RLIMIT_NOFILE, &rlp);
        getrlimit(RLIMIT_NOFILE, &rlp);

        (void) EDMDispatch_logent(
                    __FILE__,__LINE__, LOG_INFO, MESSAGE_STARTUP, 0,
                        "Service allows %d open files",
                                                    rlp.rlim_max );

        /* Initialize service launcher */
        ret = EDMDDSvcInit();

        if (ret != 0)
        {
            (void) EDMDispatch_logent(
                                __FILE__,__LINE__, LOG_INFO, 0, 0,
                        "Service Launcher failed returning - %d",
                                                            ret);
            exit(1);
        }

        /*
         * Start the other threads in the daemon. The main thread
         * becomes the RPC thread. BAMDataManage is the entry point
         * for the data collection thread. BAMDataCleanup is the
         * entry point for the data expiration thread.
         */
        /*pthread_create(&mantid, NULL, DispDaemon_ccr, NULL); */
        pthread_create(&mantid, NULL, DispDaemon_ccw, NULL);
        pthread_create(&cleantid, NULL, DispatchBackground, NULL);
        rpc_init();
        rpc_run();
}
```

```
/*********************************************************************
                                                            ********
**
** Routine: daemon_cleanup
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              None
**
** Purpose:     Call function which will clean up daemon properly.
**
** Intended caller:  internal only.
**
*********************************************************************
                                                            ********
*/

void
daemon_cleanup()
{
        kill_handler( 0 );
}
```

```
/*
** Copyright 1996,1997 EMC Corporation
*/

/* EDMDispatchService.c
 *
 *
 * Mission Statement:  RPC entry points.
 *
 * Primary Data Acted On:
 *
 * Compile-Time Options:
 *
 * Basic idea here:
 */

#if !defined(lint)
static char     RCS_id [] = "@(#)$RCSfile: EDMDispatchService.c,v $ "
                            "$Revision: 1.0 $ "
                            "$Date: 1997/02/06 20:49:15 $" ;
#endif

#include <esl/c_portable.h>
#include <esl/inout.h>

#include <logging/logging.h>
#include <csc/cscomm.h>

#include <restore/csc_EDMDispatch.h>
#include <restore/dispatch_daemon.h>

#include <EDMDispatchLog.h>
#include <EDMDispatchSession.h>

/*
 * These are all the rpc entry points for the dispatch daemon.
 * The dispatch daemon is multi-threaded and it is the main thread
 * which handles all incoming RPC. ONC RPC is single threaded
 * so each call blocks other RPC calls. This provides us some
 * safety in the way we handle our data and limits our exposure
 * to unexpected multithreading problems.
 */
static void FreeSessionInfo(SessionInfo *);

/*********************************************************************
                                                            ********
**
** Routine:  dd_initialize_1
**
** Inputs:   DD_initialize_args * - args for the restore initialize
                                                               call
**
** Outputs:  None
**
** Return Codes:
**           DD_initialize_result * - result of init function call
**
** Purpose:  Function to create a restore session.
**
** Intended caller:  Internal Only.
*********************************************************************
                                                            ********
*/

DD_initialize_result *
```

```
dd_initialize_1_svc(
                     IN DD_initialize_args *arg, IN struct svc_req *req )
{
    static DD_initialize_result argzz;

    InitializeSession(arg, req, &argzz);

    return &argzz;
}
```

```
/*********************************************************************
                                                             ********
**
** Routine:  dd_getservicestatus_1
**
** Inputs:   DD_getservicestatus_args * - args for the
                                          getservicestatus call
**
** Outputs:  None
**
** Return Codes:
**           DD_getservicestatus_result * - result of status function
                                                                 call
**
** Purpose:  Function to poll for status on a session.
**
** Intended caller:  Internal Only.
*********************************************************************
                                                             ********
*/

DD_getservicestatus_result *
dd_getservicestatus_1_svc(
          IN DD_getservicestatus_args *arg, IN struct svc_req *req )
{
    static DD_getservicestatus_result argzz;

    GetDispatchStatus(arg, &argzz);

    return &argzz;
}
```

```
/*********************************************************************
                                                             ********
**
** Routine:  dd_getsessioninfo_1
**
** Inputs:   DD_getservicestatus_args * - args for the getsessioninfo
                                                                 call
**
** Outputs:  None
**
** Return Codes:
**           SessionBlock * - result of session info call
**
** Purpose:  Function to get information on all sessions.
**
** Intended caller:  Internal Only.
*********************************************************************
                                                             ********
*/

SessionBlock *
dd_getsessioninfo_1_svc(
          IN DD_getservicestatus_args *arg, IN struct svc_req *req )
{
    static SessionBlock argzz;
    static boolean_ty first = TRUE;

    if (first)
    {
        memset(&argzz, 0, sizeof(argzz));
        first = FALSE;
    }
    else
    {
        FreeSessionInfo(argzz.sess);
        argzz.sess = NULL;
    }

    GetDispatchInfo(arg, &argzz);

    return &argzz;
}
```

```
/*********************************************************************
                                                              ********
**
** Routine:  FreeSessionInfo
**
** Inputs:   SessionInfo * - arg to free
**
** Outputs:  None
**
** Return Codes:
**           None
**
** Purpose:  Function to free all SessionInfo structures in a list.
**
** Intended caller:  Internal Only.
*********************************************************************
                                                            *********/

static void FreeSessionInfo(SessionInfo *sess)
{
    if (sess == NULL)
        return;

    if (sess -> next != NULL)
        FreeSessionInfo(sess -> next);

    free(sess);
}
```